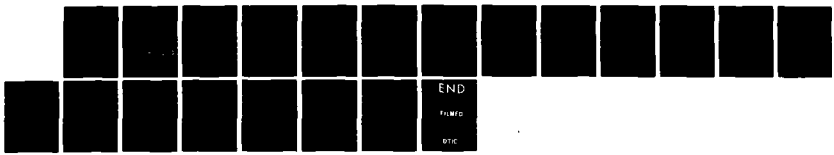
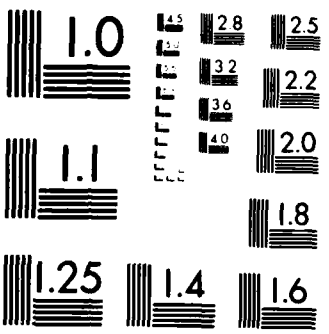
AD-A150 497 PARALLEL UPDATE OF MINIMUM SPANNING TREES IN LOGARITHMIC TIME.. (U) MARYLAND UNIV COLLEGE PARK CENTER FOR AUTOMATION RESEARCH I V RAMAKRISHNAN ET AL. NOV 84 1/1

UNCLASSIFIED CAR-TR-97 AFOSR-TR-85-0069 F/G 12/1 NL





AFOSR-TR- 85-0069

(4)

CAR-TR-97
CS-TR-1452

November 1984

PARALLEL UPDATE OF MINIMUM SPANNING TREES IN LOGARITHMIC TIME

I.V. Ramakrishnan
Shaunak Pawagi
Department of Computer Science
University of Maryland
College Park, MD 20742

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

85 8

CAR-TR-97
CS-TR-1452

November 1984

PARALLEL UPDATE OF MINIMUM SPANNING TREES IN LOGARITHMIC TIME

I.V. Ramakrishnan
Shaunak Pawagi
Department of Computer Science
University of Maryland
College Park, MD 20742

ABSTRACT

Parallel algorithms are presented for updating a minimum spanning tree when the cost of an edge changes or when a new node is inserted in the underlying graph. The machine model used is a parallel random access machine which allows simultaneous reads but prohibits simultaneous writes into the same memory location. The algorithms described in this paper for updating a minimum spanning tree require $O(\log n)$ time and $O(n^2)$ processors. These algorithms are efficient when compared to previously known algorithms for initial construction of a minimum spanning tree that require $O(\log^2 n)$ time and use $O(n^2)$ processors.

The support of the first author by the Office of Naval Research under Contract N00014-84-K-0530, and of the second author by the Air Force Office of Scientific Research under Contract F-49620-83-C-0082, is gratefully acknowledged.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): CAR-TR-97; CS-TR-1452	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 85-0069
6a. NAME OF PERFORMING ORGANIZATION: University of Maryland; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research
6c. ADDRESS (City, State and ZIP Code): Department of Computer Science, College Park MD 20742	7b. ADDRESS (City, State and ZIP Code): Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR; 8b. OFFICE SYMBOL (If applicable): NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-83-C-0082
8c. ADDRESS (City, State and ZIP Code): Bolling AFB DC 20332-6448	10. SOURCE OF FUNDING NOS. — PROGRAM ELEMENT NO.: 61102F; PROJECT NO.: 2304; TASK NO.: A7; WORK UNIT NO:

11. TITLE (Include Security Classification): PARALLEL UPDATE OF MINIMUM SPANNING TREES IN LOGARITHMIC TIME

12. PERSONAL AUTHOR(S): I.V. Ramakrishnan and Shaunak Pawagi

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	NOV 84	16

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Parallel algorithms are presented for updating a minimum spanning tree when the cost of an edge changes or when a new node is inserted in the underlying graph. The machine model used is a parallel random access machine which allows simultaneous reads but prohibits simultaneous writes into the same memory location. The algorithms described in this paper for updating a minimum spanning tree require $O(\log n)$ time and $O(n^2)$ processors. These algorithms are efficient when compared to previously known algorithms for initial construction of a minimum spanning tree that require $O(\log^2 n)$ time and use $O(n^2)$ processors.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Dr. Robert N. Buchal	(202) 767- 4939	NM

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

1. Introduction

Incremental graph algorithms deal with recomputing properties of graph after an incremental change is made to the graph, such as addition and deletion of vertices and edges, as well as changes in the costs or capacities (if any) associated with the edges of the graph. Such recomputations are also referred to as "updating" graph properties.

Incremental graph algorithms have received considerable attention in the past. In particular, Spira and Pan [12], Chin and Houck [2] and Frederickson [6] have investigated the update problem for minimum spanning tree (MST) of an undirected graph. Cheston [1] and Spira and Pan [12] describe the algorithms for shortest path update. Even and Shiloach [3] have investigated the update of connected component problem under the operation of edge deletion. Ibaraki and Katoh [8] examine incremental algorithms for the transitive closure of a directed graph.

The problem of updating an MST involves reconstructing the new MST from the current MST when the cost of an edge has changed or a vertex along with all its incident edges is inserted or deleted from the underlying graph. We refer to these two subproblems as the edge update and the vertex update problem respectively. Frederickson [6] describes an $O(\sqrt{m})$ algorithm for the edge update problem, where m is the number of edges in the graph. Spira and Pan [12] and Chin and Houck [2] present an O(n) algorithm for updating the MST of an n vertex graph when a new vertex is inserted into the graph.

Parallel algorithms for updating an MST have not been studied so far. In this paper we present parallel algorithms for updating an MST. Our model of computation is the single instruction multiple data stream (SIMD) model. We assume that all processors have access to a common memory and that simultaneous reads from the same loca-

A-1

tion are allowed but simultaneous writes to the same location are prohibited. Fortune and Wylie [5] call such model a parallel random access machine (PRAM). Savage and Ja'Ja' [11] and Chin et al. [3] have described an $O(\log^2 n)$ ** algorithm for constructing an MST on PRAMs. Our algorithm for the edge update problem requires O(log n) time. By using a novel approach to reconstruct an MST we also solve the vertex insertion problem in O(log n) time.

The rest of the paper is organized into four sections. In Section 2 we describe some graph-theoretic preliminaries adopting the framework in [13]. In Section 3 we describe the edge update algorithm, and the vertex insertion algorithm is described in Section 4.

2. Preliminaries

Let G=(V,E) denote a *graph* where V is a finite set of vertices and E is a set of pairs of vertices called edges. If the edges are unordered pairs then G is *undirected* else it is *directed*. Throughout this paper we assume that $V=\{1,2,\ldots,n\}$, $|V|=n$ and $|E|=m$. We denote the undirected edge from a to b by (a,b) and the directed edge between them by $<a,b>$. We say that an undirected graph G is *connected* if for every pair of vertices u and v in V, there is a path in G joining u and v. Each connected maximal subgraph of G is called a *component* of G. An adjacency matrix A of G is an $n \times n$ Boolean matrix such that $A[u,v]=1$ if and only if $(u,v) \in E$. A *tree* is a connected undirected graph with no cycles in it. Let $T=(V',E')$ be a directed graph. T is said to have a *root* r, if $r \in V'$ and every vertex $v \in V'$ is reachable from r via a directed path. If the underlying undirected graph of T is a tree then T is called a directed tree. If the edges of T are all reversed then the resulting graph is called an *inverted* tree. We denote the "undirected"

** Throughout this paper, we use log n to denote $\lceil \log_2 n \rceil$.

path from vertex a to vertex b by [a-b] and directed path by [a→b]. Let T be a directed tree with $u,v \in V'$. Then the *lowest common ancestor* (LCA(u,v)) of u and v in T is the vertex $w \in V'$ such that w is a common ancestor of u and v, and any other common ancestor of u and v in T is also an ancestor of w in T. Let C:E→R denote a function that associates a cost with the edges of G. A minimum spanning tree of G is a spanning tree of G such that sum of the costs of the edges in the tree is minimum over all spanning trees for G.

As we will see later on, our algorithm for updating an MST (vertex insertion in particular) requires the paths from all vertices to the root in an inverted tree. Tsin and Chin [13] have described a technique due to Savage [10] to compute all such paths. For completeness we now describe their technique.

Let $T=(V',E')$ be an inverted tree with $V'=\{1,2,...,n\}$ and $|V'|=n$. Let r be the root of this tree. For a directed edge $<a,b>$ we say that vertex b is the father of vertex a.

Definition: $F:V' \rightarrow V'$ is a function such that F(i)=the father of vertex i in T for $i \neq r$ and F(r)=r.

The function F can be represented by a directed graph F which can be constructed from T by adding a self-loop to the root r.

From the function F, we define F^k, $k \geq 0$ as follows.

Definition: $F^k:V' \rightarrow V'$ $(k \geq 0)$ such that $F^0(i)=i$, for all $i \in V'$ and $F^k(i)=F(F^{k-1}(i))$, for all $i \in V'$ and $k>0$.

If i is a vertex in T, $F^k(i)$ is the k^{th} ancestor of i in the inverted tree.

Definition: For each $i \in V'$, $depth(i)=\min\{k|F^k(i)=r \text{ and } 0 \leq k < n\}$.

Lemma 2.1: Given the function F of an inverted tree, F^k can be computed in O(log n) time using $O(n^2)$ processors.

Proof: To compute F^k $(0 \leq k < n)$ we proceed as follows. We assume that the processors are indexed as P(1,1), P(1,2),..., P(n,n). The instructions within "pardo...dopar" are executed in parallel and comments are enclosed within //...//.

1. for all i $(1 \leq i \leq n)$ pardo $F^0(i)=i$, $F^1(i)=F(i)$ dopar; //Processor P(1,i) executes the instruction within pardo...dopar.//

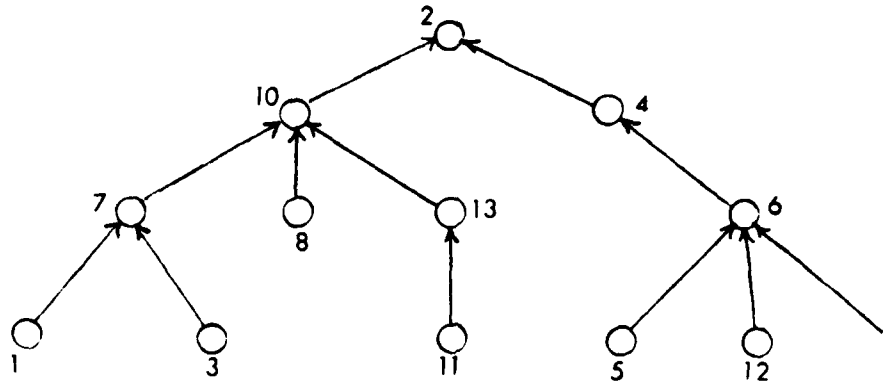
2. for t:=0 to log(n-1)-1 do

   for all s $(1 \leq s \leq 2^t)$ and for all i $(1 \leq i \leq n)$ pardo $F^{2^t+s}:=F^{2^t}(F^s(i))$ dopar; //Processor P(s,i) executes the instruction within pardo...dopar.//

Now step (1) can be done in constant time using n processors. To do the i^{th} iteration of step (2) in constant time we require $2^i n$ processors. As there are log(n-1)-1 iterations of step (2), we therefore require $O(n^2)$ processors.

The actual computations of $F^k(i)$ $(1 \leq i \leq n, 1 \leq k < n)$ are performed in an array F^+ in which $F^+[i,k]$ contains $F^k(i)$. Once the F^+ array is computed, depth(i) $(1 \leq i \leq n)$ can be found by performing a binary search on the i^{th} row. We search for the left-most occurrence of r. This takes log n time by assigning a processor per row. However, it can be done in constant time by assigning a processor to each element in F^+. This is done as follows. Every processor compares its element with the elements in its left and right neighbors. There is exactly one processor which does not have all the three elements identical or distinct and this processor locates the left-most occurrence of r. The depth information is stored in a one-dimensional array D^+.

After the computations for D^+ are finished, each row of F^+ is right shifted so that all the r's except the left-most one are eliminated. As a consequence, the right-most column of the array contains only the root r. Fig. 2.1 below illustrates an inverted tree and its array F^+ after the rows have been shifted right.



	0	1	2	3	4	5	6	7	8	9	10	11	12
1										1	7	10	2
2													2
3										3	7	10	2
4												4	2
5										5	6	4	2
6											6	4	2
7											7	10	2
8											8	10	2
9										9	6	4	2
10												10	2
11										11	13	10	2
12										12	6	4	2
13											13	10	2

Undefined entries are left blank.

Fig 2.1

Lemma 2.2: We can compute the lowest common ancestors of ${}^nC_2$ vertex pairs in the inverted tree in O(log n) time using $O(n^2)$ processors.

Proof: We make use of the array F^+ to design a parallel algorithm for finding the lowest common ancestors. Let a and b be a vertex pair. If c is their lowest common ancestor, then row a and row b of F^+ will have identical contents for column n-1, column n-2,..., down to the column containing c. After this column the contents of rows a and b differ. As a result, to determine c, we can perform a binary search on row a and row b simultaneously in the following way. If the two entries being examined in row a and row b (in the same column) are different, the search is continued on the right half, otherwise it is continued on the left half. It takes (log n)+1 time steps to find c with one processor. □

Having obtained the lowest common ancestor we can now identify the unique path between any two vertices (passing through their lowest common ancestor). We now describe how to compute the maximum cost edge on the unique path between any two vertices.

Let $E_m(e_1,e_2)$ denote the maximum cost edge between edges e_1 and e_2. Let $F_m^k(i)$ $(1 \leq i \leq n)$ be the maximum cost edge on the path from i to its k^{th} ancestor in T. Then

1. $F_m^1(i)$ is the edge $(i,F^1(i))$
2. $F_m^k(i)$ is the edge $E_m(F_m{}^k(i),(F^{k-1}(i),F^k(i))$, $k>1$.

We assume that the cost of the edge (r,r) in T is $-\infty$.

Lemma 2.3: We can compute $F_m^k(i)$ for all i $(1 \leq i \leq n)$ in O(log n) time using $O(n^2)$ processors.

Proof: We describe an algorithm to compute $F_m^k(i)$.

1. For all i ($1 \leq i \leq n$) pardo $F_m^1(i)=(i,F^1(i))$ dopar;

2. For t:=0 to log(n-1)-1 do

   For all s ($1 \leq s \leq 2^t$) and for all i ($1 \leq i \leq n$) pardo $F_m^{2^t+s}(i)=E_m(F_m{}^{2^t}(i),\ F_m^s(F^{2^t}(i)))$ dopar;

The analysis is similar to the proof of Lemma 2.1. □

The computations of $F_m^k(i)$ are done in a two-dimensional array F_m^+. $F_m^+(i,k)$ is the maximum cost edge on the path from i to its k^{th} ancestor.

Lemma 2.4: Given F^+, E_m^+, and D^+, we can find the maximum cost edge on the path [u-v] (for all $u,v \in V'$) in O(log n) time using $O(n^2)$ processors.

Proof: First we find LCA(u,v). By Lemma 2.2, this can be done in O(log n) time using $O(n^2)$ processors. Let p be the LCA(u,v). Let $x=D^+[u]-D^+[p]$ and $y=D^+[v]-D^+[p]$ (that is, p is the x^{th} ancestor of u and the y^{th} ancestor of v). Finding x and y for all pairs u,v takes constant time using $O(n^2)$ processors. Finally, the maximum cost edge on the path [u-v] is $E_m(F_m^x(u),\ F_m^y(v))$. This again can be computed for all pairs u,v in constant time using $O(n^2)$ processors. □

The maximum cost edge on the path [u-v] for all pairs u,v is stored in a two-dimensional array M^+. Note that if F represents a forest of inverted trees then F^+, D^+ and M^+ contain information about ancestors, depth and maximum cost edges for all the trees in the forest.

3. Edge Update Algorithms

The edge update problem is concerned with reconstructing the new MST when the cost of an edge in the underlying graph changes. There are several cases to be handled in edge-cost updating. The cost of an edge may either increase or decrease and this edge may currently be either in the tree or not in the tree. If the cost of a tree edge decreases, or the cost of a non-tree edge increases, then the old MST will not undergo any change. On the other hand, it may undergo changes when the cost of a tree edge increases or the cost of a non-tree edge decreases. However, in both cases, at most one edge will enter the tree and another edge will leave it.

If the cost of a tree-edge (x,y) increases then the new MST is recomputed as follows.

1. Delete the tree-edge (x,y). This creates a forest of two subtrees.
2. Identify the vertices in each of these subtrees.
3. Find the minimum cost edge connecting them.

If the cost of a non-tree edge (u,v) decreases, then we proceed to recompute the new MST as follows.

1 Add (u,v) to the old MST. The edge (u,v) induces a cycle in the old MST.
2. Remove the maximum cost edge on this cycle.

See Chin and Houck [2] for a proof of correctness of both these algorithms.

We assume that the update algorithms operate on an MST in the form of an inverted tree (see Section 2) with an arbitrary vertex as the root. After an edge update, the algorithms ensure that the reconstructed MST is also preserved as an inverted tree. Using the technique of Tsin and Chin [13], the parallel algorithm for constructing the

MST in [3.11] can be easily modified to yield an MST in the form of an inverted tree. Such an algorithm requires $O(\log^2 n)$ time and uses $O(n^2)$ processors.

We now describe a parallel algorithm to update the MST when the cost of a tree edge (x,y) increases. Let r be the root of the inverted MST. The steps are as follows.

1. We assume, without loss of generality, that the direction of edge (x,y) is from x to y. Now set $F^1(x)=x$. This deletes the directed edge $<x,y>$ from the inverted MST and creates a forest of two subtrees, one of which is rooted at r and the other at x. This step can be done in constant time with a single processor.

2. Compute the array F^+. By Lemma 2.1, this can be done in O(log n) time using $O(n^2)$ processors. At the end of this step, all vertices in the subtree rooted at r will have r in their last column in F^+ and all vertices in the subtree rooted at x will have x in their last column. We therefore can identify the vertices in the two subtrees.

3. Determine the minimum cost edge connecting these two subtrees. This involves the following steps.

   3a. For each vertex i find the minimum cost edge (i,j) such that i and j are not in the same subtree. Since there are at most n edges incident on i, step (3a) can be done in O(log n) time using $O(n^2)$ processors, by assigning n processors to each vertex.

   3b. The minimum cost edge connecting these two subtrees can now be found by selecting the minimum cost edge among the edges selected in step (3a). As there are at most n edges, such a selection can again be done in O(log n) time using O(n) processors.

4. Let (u,v) be the edge selected in step (3). If edge (u,v) is the same as edge (x,y) then let $F^1(x)=y$ (that is, the old MST does not change). On the other hand, if edge (u,v) is not the same as edge (x,y) then the two subtrees and the edge (u,v) form the new MST.

5. Finally, we must maintain the new MST is an inverted tree. To do so, we proceed as follows.

   Assume, without loss of generality, that u is in the subtree rooted at x and v is in the subtree rooted at r. Now orient the edge (u,v) from u to v. To do so set $F^1(u) = v$. In step (2) we found the path from vertex v to x. Now reverse the directions of the edges on the directed path $|v \rightarrow x|$ in the old inverted MST. For instance, if the directed edge $<a,b>$ was on the directed path $|v \rightarrow x|$ then set $F^1(b) = a$. This path can have at most n edges and hence the reversal can be done in constant time using O(n) processors.

This completes the description of the parallel algorithm to update the MST when the cost of a tree edge (x,y) increases. We now describe a parallel algorithm to update the MST when the cost of a non-tree edge (u,v) decreases. Again, let r be the root of the inverted MST. The steps then are as follows.

1. Compute arrays F^t, F_m^t and D^t. By Lemma 2.4, we can find the maximum cost edge (x,y) on the path |u-v| in the MST is $O(\log n)$ time using $O(n^2)$ processors.

2. If the cost of edge (x,y) is less or equal to the cost of edge (u,v) then the old MST does not change. Otherwise, the edge (x,y) must be deleted from the MST and edge (u,v) must be added.

3. Assume, without loss of generality, that the direction of edge (x,y) in the inverted MST is from x to y. Now set $F^1(x)=x$ and compute F^t. If u is in the subtree

rooted at x then direct edge (u,v) from u to v, else direct it from v to u. Computing F^+ takes O(log n) time using $O(n^2)$ processors.

4. Finally, we have to maintain the new MST as an inverted tree. This can be done in constant time using O(n) processors (see step (5) of the previous algorithm).

Note that edge insertion and edge deletion can be easily handled by our algorithms. Assign large positive costs ($+\infty$) to edges not in the underlying graph. If such an edge is inserted into the graph then we can consider this as equivalent to decreasing the cost of a non-tree edge. Similarly edge deletion from the MST can be handled by again assigning a large positive cost to that edge and this in turn is equivalent to increasing the cost of a tree edge.

4. Vertex Update Algorithm

The vertex update problem involves reconstructing the new MST when a vertex is either inserted or deleted from the underlying graph. We now describe our method of handling the vertex update problem when a new node is inserted into the underlying graph. The other case of reconstructing the MST when a vertex is deleted from the graphs appears difficult to handle. For instance, if the MST is in the form of a "star" (that is, there exists a vertex on which all the edges in the MST are incident), the deletion of such a vertex deletes all the edges in the tree. Updating the MST then requires reconstructing it all over again (that is, by examining all the remaining edges in the graph).

Spira and Pan [12] update the MST in O(n) time when a vertex is inserted in the graph. Their algorithm constructs the MST all over again by examining the n-1 edges in the old MST and the new edges (there can be at most n of them) brought in by the

inserted vertex. The best sequential algorithm for constructing an MST requires $O(n^2)$ time for dense graphs [9] and O(m loglog n) time for sparse graphs [14] (recall that m is the number of edges in the graph). The O(n) time complexity obtained by Spira and Pan to update the MST is primarily due to the smaller number of edges that need to be examined. However, parallel algorithms to construct an MST [3,11] by just examining the edges in the old MST and the new edges brought in by the inserted vertex still requires $O(\log^2 n)$ time. Chin and Houck [2] also describe a sequential algorithm of time complexity O(n) for the vertex update problem when a new vertex is inserted in the graph. Their algorithm, however, is inherently sequential.

Our solution to this problem requires a novel way of examining the old tree edges and the new edges brought in by the inserted vertex. Every pair of edges incident on the new vertex induces a cycle in the old MST. At most n such edges are incident on the inserted vertex thereby creating ${}^nC_2$, that is $O(n^2)$ cycles. We break all these cycles *simultaneously* by *removing* the *maximum-cost* edge on each cycle. We show later on that the resulting graph is a minimum spanning tree. The details of our algorithm are as follows.

Let z be the new vertex inserted in the graph.

1. Compute array M^+ for the old MST. By Lemma 2.4, this can be done in O(log n) time using $O(n^2)$ processors.

2. Find out the maximum cost edge on each cycles induced by z in the old MST. For instance, let u and v be any two vertices in the MST and let (z,u) and (z,v) be the two new edges incident on u and v. Now $M^+(u,v)$ is the maximum cost edge on the path [u-v]. The maximum cost edge on the cycle formed by the edge (z,u), the path [u-v] and the edge (v,z) is obtained by selecting the maximum cost edge among the

edges (z,u), (v,z) and $M^+[u,v]$ (recall that $M^+[u,v]$ is the maximum cost edge on the path [u-v] in the MST). This selection can be done in constant time using $O(n^2)$ processors.

3. Delete the maximum cost edges selected in step (2). For instance, let (x,y) be such an edge. Assume, without loss of generality, that its direction in the inverted tree is from x to y. Then, to delete this edge from the inverted tree set $F^1(x)=x$. This is done by setting $F^+[x,1]=x$ in the F^+ array. Since an edge may be selected for deletion by more than one processor, a write conflict may arise. However, such a conflict can be avoided by using the buddy-system technique due to Hirschberg [7]. This technique deletes all the selected edges without write conflicts in O(log n) time using $O(n^2)$ processors.

4. Finally, we must maintain the new MST as an inverted tree. To do so, we proceed as follows. The subtrees created by the deletion of the edges in the old MST are now connected to each other through edges incident on z. Let $x_1,x_2, \ldots, x_k$ be the roots of the k such subtrees formed in step (3). Let $w_1,w_2, \ldots, w_k$ be the vertices in the subtrees rooted at $x_1,x_2, \ldots, x_k$ respectively that have edges incident on z. Compute array F^+. This array contains the paths from w_1 to x_1, w_2 to $x_2,\ldots,w_k$ to x_k. Now reverse the direction of all the edges on these paths. Next, orient all the edges (w_1,z), $(w_2,z),\ldots,$ (w_k,z) towards z. Thus, z becomes the root of the inverted tree representing the new MST. By Lemma 2.1, computation of the F^+ array requires O(log n) time using $O(n^2)$ processors. Also, reversal of the edges can be done in constant time using $O(n^2)$ processors.

This completes the description of the algorithm. We will now show that our algorithm indeed produces an MST.

Theorem 4.1: Our algorithm computes the new MST after a vertex insertion.

Proof: Let T' be the graph obtained after steps (1), (2), (3) and (4) of our algorithm are executed.

First, T' is acyclic as all the cycles are broken in step (3) of our algorithm. We next show that T' is connected. Consider a vertex u in T' and let $e_1, e_2, \ldots, e_k$ be the edges incident on it (see Fig. 4.1).

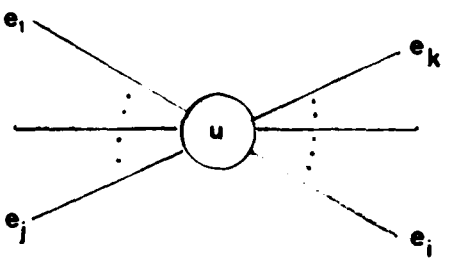


Fig 4.1

Let edge e_j ($j \leq k$) be the minimum cost edge among them. Now consider all the cycles containing e_j that pass through u. They must also contain some other e_l ($e_l \neq e_j$). When these cycles are broken in step (3) of our algorithm e_j is retained as it is the minimum cost edge incident on u.(If at all any edge incident on u is selected for deletion for cycles passing through u, it is not e_j .) Therefore the minimum cost edge incident on each vertex is retained. This in turn creates at most $\frac{n}{2}$ components.

Now, assume that there are k ($k > 1$) such components. Using a similar argument based on cycles passing through each component it is easy to see that the minimum cost edge incident on each component is retained. This in turn creates at most $\frac{k}{2}$ components. As the number of components monotonically decrease, we are eventually left with one component. Therefore T' is connected. T' is a tree as it is acyclic and con-

nected.

Finally, it is well known that every non-tree edge is the maximum cost edge on the cycle it induces in the MST. Our algorithm deletes the maximum cost edge on each cycle induced by z. Hence T' is an MST. □

Theorem 4.2: Our algorithm takes O(log n) time and requires $O(n^2)$ processors.

Proof: Steps (1), (2), (3) and (4) of our algorithm. □

5. Conclusions

Incremental graph algorithms deal with recomputing properties of a graph after an incremental change has been made to the graph. In this paper we have examined the problem of updating a minimum spanning tree. We have described a parallel algorithm to update an MST when the cost of an edge changes or a new node is inserted in the underlying graph. Our algorithm requires O(log n) time using $O(n^2)$ processors. It is therefore efficient when compared to parallel algorithms for initial construction of a minimum spanning tree which take $O(\log^2 n)$ time and use $O(n^2)$ processors.

References

[1] Cheston, G., *Incremental Algorithms in Graph Theory*, TR 91, Dept. of Computer Science, Univ. of Toronto (1976).

[2] Chin, F. and Houck, D., *Algorithms for Updating Minimum Spanning Trees*, J. Comp. Syst. Sci., 16 (1978), pp. 333-344.

[3] Chin, F., Lam, J. and Chen, I., *Efficient Parallel Algorithms for Some Graph Problems*, Comm. ACM, 25 (1982), pp. 170-175.

[4] Even, S. and Shiloach Y., *An On-line Edge Deletion Problem*, J. ACM, 28 (1982), pp. 1-4.

[5] Fortune, S. and Wyllie, J., *Parallelism in Random Access Machines*, Proc. Tenth Symposium on Theory of Computing, San Diego (1978), pp. 114-118.

[6] Frederickson, G., *Data Structures for On-line Updating of Minimum Spanning Trees*, Proc. Fifteenth ACM Symposium on Theory of Computing, Boston (1983), pp. 252-257.

[7] Hirschberg, D., *Fast Parallel Sorting Algorithms*, Comm. ACM, 21 (1978), pp. 657-661.

[8] Ibaraki, T. and Katoh, N., *On-line Computation of Transitive Closure of Graphs* , Inf. Proc. Letters, 16 (1983), pp. 95-97.

[9] Prim, R., *Shortest Interconnection Networks and Some Generalizations*, Bell System Tech. J., 36 (1967), pp. 1389-1401.

[10] Savage, C., *Parallel Algorithms for Some Graph Problems*, TR-784, Dept. of Mathematics, Univ. of Illinois, Urbana (1977).

[11] Savage, C. and Ja'Ja' J., *Fast Efficient Parallel Algorithms for Some Graph Problems*, SIAM J. Comp., 10 (1981), pp. 682-691.

[12] Spira, P. and Pan. A., *On Finding and Updating Spanning Trees and Shortest Paths*, SIAM J. Comp., 1 (1972), pp. 146-160.

[13] Tsin, Y. and Chin, F., *Efficient Parallel Algorithms for a Class of Graph Theoretic problems*, SIAM J. Comp., 14 (1984), pp. 580-599.

[14] Yao, A., *An $O(|E|loglog|V|)$ Algorithm for Finding Minimum Spanning Trees*, Inf. Proc. Letters, 4 (1975), pp. 21-23.

END

FILMED

3-85

DTIC